Asymptotically-Optimal Multi-Query Path Planning for a Polygonal Robot

Duo Zhang Zihe Ye Jingjin Yu

***Abstract*— Shortest-path roadmaps, also known as reduced visibility graphs, provide a highly efficient multi-query method for computing optimal paths in two-dimensional environments. Combined with Minkowski sum computations, shortest-path roadmaps can compute optimal paths for a translating robot in 2D. In this study, we explore the intuitive idea of stacking up a set of reduced visibility graphs at different orientations for a polygonal holonomic robot to support the fast computation of near-optimal paths, allowing simultaneous 2D translation and rotation. The resulting algorithm, *rotation-stacked visibility graph* (RVG), is shown to be resolution-complete and asymptotically optimal. Extensive computational experiments show RVG significantly outperforms state-of-the-art single- and multi-query sampling-based methods on both computation time and solution optimality fronts. Source code and supplementary materials are available at https://github.com/arc-l/rvg.**

I. Introduction

In many robotics applications, it is desirable to compute high-quality collision-free paths quickly for a translating and rotating object in two dimensions. A prominent example is the path planning for robotic vehicles, which are often rectangular (including squares), to navigate in indoor (e.g., warehouses) or outdoor environments. Another frequently encountered challenge is finding such paths for manipulating objects on flat surfaces. This can happen when heavy furniture is being moved around in a house or when some objects to be rearranged on a tabletop cannot be lifted above other objects and directly transported around (e.g., [1], see Fig. 1). In these cases, a short (in terms of Euclidean distance) collision-free path is often preferred. Whereas such problems can be solved using combinatorial approaches [2]–[5], sampling-based algorithms [6]–[14] and gradient-based methods [15]–[20], either solution quality could be improved or computation time needs to be accelerated. In addition, the time required to reach a solution can vary greatly for random sampling-based methods. This raises a natural question: Can a specialized high-performance algorithm be developed to plan high-quality paths for moving a polygonal robot in 2D, amongst polygonal obstacles?

Shortest-path roadmaps [21], also called *reduced visibility graphs* [22], find the shortest path connecting two points in two dimensions where bitangents can be computed for the obstacles (including the environment boundary). Leveraging shortest-path roadmaps, it is shown [23] that 2D shortest paths can be effectively computed for translating a polygon among static polygonal obstacles. This is achieved by first explicitly computing the *configuration space* or $\mathcal{C}$-space [24] of the movable polygon by taking the Minkowski sum [25] between the polygon and the environment. Then, a shortest-path roadmap can be computed over the resulting free configuration space $\mathcal{C}_{free}$, allowing arbitrary shortest-path queries between two points within $\mathcal{C}_{free}$ to be quickly resolved. In the same study [23], 2D rotations are also considered to a limited extent through combining the start and goal rotations, which is sub-optimal and incomplete. It is hypothesized, but not explored further, that dividing the rotational degree of freedom can achieve a better outcome.

D. Zhang, Z. Ye, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. Emails: {duo.zhang, zihe.ye, jingjin.yu}@rutgers.edu.

This work was supported in part by NSF awards IIS-1845888, IIS-2132972, and CCF-2309866.

Fig. 1: A motivating example in developing RVG. In object rearrangement, high-quality paths often need to be planned to translate and rotate objects in a 2D workspace, e.g., a tabletop. Images are reproduced from [1].

Motivated by vast applications and inspired by [23], in this work, we develop *rotation-stacked reduced visibility graph* ($\mathcal{RVG}$) for computing asymptotically-optimal, collision-free paths for moving a polygon (equivalently, a holonomic robot with a polygonal footprint) in two dimensions. Roughly speaking, RVG (our method for computing $\mathcal{RVG}$) slices the rotational degree of the $SE(2)$ configuration space. After computing the Minkowski sum of a rotated polygon with the environment for each rotation slice, the corresponding reduced visibility graph is constructed. Then, RVG carefully connects these reduced visibility graphs to yield a single rotation-stacked reduced visibility graph. We prove that RVG is resolution-complete and its solutions converge to the optimal solution in the limit. Perhaps more importantly, we show that RVG, as a multi-query method, delivers better computation time and solution optimality than state-of-the-art single- and multi-query sampling-based methods.

II. Preliminaries

A. Problem Definition

Let $\mathcal{W} \subset \mathbb{R}^2$ be a compact (i.e., closed and bounded) workspace. There is a set of polygonal obstacles $\{O_i\}$. For all i, $O_i \subset \mathcal{W}$ is compact. For $i \neq j$, $O_i \cap O_j = \varnothing$. Let $O = \bigcup_{O_i \in \{O_i\}} O_i$. A polygonal holonomic robot r resides in the $\mathcal{W} \backslash O$, assuming a configuration $q = (x, y, \theta) \in SE(2)$. Let the free configuration space for the robot be $\mathcal{C}_{free}$ and let its closure be $\overline{\mathcal{C}_{free}}$. A *feasible path* for the robot is a continuous map $\tau : [0, 1] \rightarrow \overline{\mathcal{C}_{free}}$. Intuitively, the robot may touch obstacles but may not penetrate any obstacles.

We make a general position assumption that the robot may not be "sandwiched" between two obstacles in $SE(2)$, simultaneously touching both. Formally, for any feasible path τ that touches the boundary of $\overline{\mathcal{C}_{free}}$, there exists another feasible path τ' in the same homotopy class as τ such that for all $q \in \tau'$, the $\delta = (\delta_x, \delta_y, \delta_\theta)$ open ball around q for some arbitrarily small but fixed $\delta_x, \delta_y, \delta_\theta > 0$, denoted as $B_\delta(q)$, is contained in $\mathcal{C}_{free}$. We call this the *δ-clearance* property. This suggests $B_\delta(\tau(0))$ and $B_\delta(\tau(1))$ must themselves be contained in $\mathcal{C}_{free}$, which is a very mild assumption.

A *cost metric* is needed to compute the cost of feasible paths to measure path optimality. Generally, such a cost for $SE(2)$ depends on weights given to the rotational degree. To that end, we define the cost of a path τ as

$$J(\tau) = \alpha \int_0^1 \sqrt{dx^2 + dy^2} + \beta \int_0^1 |d\theta|, \qquad (1)$$
$$\alpha, \beta \geq 0,\ \alpha\beta \neq 0,$$

where x, y, θ are functions of t. In other words, the cost of a path τ is the weighted sum of the Euclidean path length of τ's projection to $\mathbb{R}^2$ and the cumulative rotations. The $\alpha = 1$ and $\beta = 0$ case corresponds to ignoring the rotational cost. Note that costs other than Eq. (1) may be defined depending on the robot's physical constraints.

Given two configurations $q_0, q_1 \in \mathcal{C}_{free}$ between which a feasible path exists, let $\tau*$ be the path with $\tau^*(0) = q_0$ and $\tau^*(1) = q_1$ such that $J(\tau*)$ is minimized, i.e., $J^*_{q_0,q_1} := J(\tau^*) = \inf_\tau J(\tau)$. This work seeks to compute a path τ with its cost $J(\tau)$ asymptotically approaching $J^*_{q_0,q_1}$.

B. Visibility and Visibility Graphs

We briefly introduce concepts surrounding visibility and visibility graphs. For two-dimensional points $p_1, p_2 \in \mathcal{W} \backslash O$, they are *visible* to each other if $p(t) = tp_1 + (1-t)p_2 \in \mathcal{W} \backslash O$ for all $t \in [0,1]$. The definition may be naturally extended to any configuration space, including $SE(2)$: for two configurations $q_1, q_2 \in \mathcal{C}_{free}$, they are mutually visible if $q(t) = tq_1 + (1-t)q_2 \in \mathcal{C}_{free}$ for all $t \in [0,1]$. A *visibility graph* $\mathcal{VG}(V, E)$ can then be readily defined based on the definition of visibility, which contains a discrete set of vertices $V \subset \mathcal{C}_{free}$. For any two vertices $v_1, v_2 \in V$, if they are visible to each other, then there exists a straight edge $(v_1, v_2) \in E$. Conversely, E only contains such edges.

Visibility graphs, properly computed over some free configuration space, greatly facilitate the computation of optimal paths between configurations that are not mutually visible. For computing optimal paths, considering all configuration space features is often unnecessary. For example, in two-dimensional polygonal environments (Fig. 2), only *reflex vertices* (i.e., vertices at which the angle is larger than 180 degrees) are needed, and only bitangents between reflex vertices are needed as the edges. Such visibility graphs are known as *reduced visibility graphs*.

III. Algorithms

For a given (orientation) resolution n, RVG slices $SO(2)$ into n layers of uniform thickness, parameterized by feasible rotation interval $[\theta_{lb}, \theta_{ub}]$. Subsequently, each $\mathcal{L}ayer$ is built with its visibility graph $\mathcal{VG}(V, E)$ by Alg. 2, allowing navigation with free rotation within $[\theta_{lb}, \theta_{ub}]$. Each vertex $v(x, y, [\theta_{lb}, \theta_{ub}])$ in V represents a feasible subspace of $SE(2)$ consisting of a feasible position in $\mathbb{R}^2$ and a feasible range of rotations $[\theta_{lb}, \theta_{ub}]$ in $SO(2)$. To enable the robot to rotate across layers, in Alg. 3, we propagate vertices in each $\mathcal{VG}$ whose feasible rotation range is larger than $[\theta_{lb}, \theta_{ub}]$ to neighboring layers by connecting such vertices to the vertices in the $\mathcal{VG}$s located in neighboring layers. After propagation, $\mathcal{VG}$s in all layers are merged into an $\mathcal{RVG}$. With the $\mathcal{RVG}$, shortest paths can be found by any path-finding algorithm, such as A*, given any start and goal pairs.

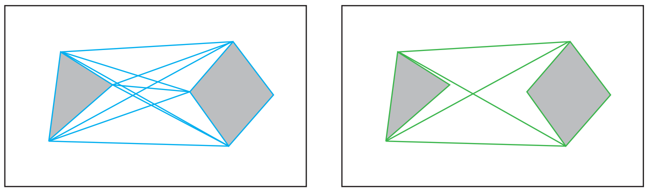

Fig. 2: [Left] A 2D polygonal environment and the (blue) visibility graph for it. [Right] The (green) reduced visibility graph for the same environment. The additional blue edges do not appear in shortest paths unless the start/goal happens to fall on them (a zero probability event).

A. Building Layers

1) Visibility Queries: To build the $\mathcal{VG}$ for a layer, for a reflex vertex v, all vertices visible to v must be retrieved. We use the TriangleExpansion algorithm [26] to determine the region visible to v given v and $\{O_i\}$.

2) Build the Visibility Graph: Given a robot geometry P_r, we can find the bounding polygon P of the rotation range when the robot is rotated to θ_{lb} and θ_{lb} respectively. Note that P needs to be an overestimate to ensure the result is collision-free and needs to converge to the true rotation range when the resolution goes to infinity. All bounding polygons are merged into one afterward. Then, the Minkowski differences between P and the obstacles $\{O_i\}$ are taken as the grown obstacles $\{O'_i\}$, as shown in Fig 3, after which the $\mathcal{VG}$ is constructed on the vertices of grown obstacles $\{O'_i\}$. The visible regions of all vertices are cached in a dictionary C for future use. Alg. 2 outlines the whole process.

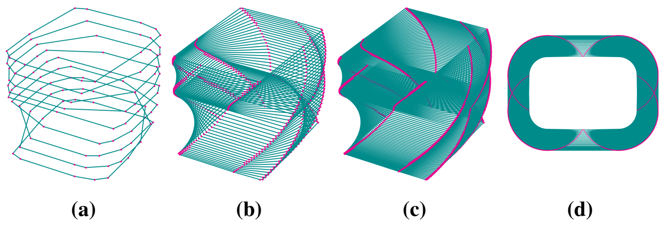

Fig. 3: (a)-(c) The semi-algebraic sets composed of the grown obstacles from all layers with resolutions=18, 90, 180 from left to right resulted from 2 rectangles, a robot and an obstacle, shown in Fig 5. (d) The semi-algebraic sets projected to the $\mathbb{R}^2$ space when resolution=180.

B. Vertex Propagation

In a $\mathcal{VG}$, a random vertex v can rotate within the range $[\theta_{lb}, \theta_{ub}]$ if it is not inside the grown obstacles. Thus, if a vertex is not located in any of the grown obstacles in the neighboring layers, it can freely rotate in the range of neighboring layers at exactly its own location in $\mathbb{R}^2$.

Algorithm 1: VISIBILITYQUERY($v, \{O_i\}$)

```
1 $V = \varnothing$, $A = $ TRIANGLEEXPANSION($v, \{O_i\}$)
2 for $O_i \in \{O_i\}$ do
3     for $v' \in O_i$ do
4         if $v' \in A$ then $V = V \cup v'$ ;
5 return $V, A$
```

Algorithm 2: BUILDLAYER($P_r, [\theta_{lb}, \theta_{ub}], \{O_i\}$)

```
1 find the bounding polygon $P$ when $P_r$ is rotated to $\theta_{lb}$ and $\theta_{ub}$ respectively
2 move $P$ to the origin, invert $P$
3 $\{O'_i\} = \{P \ominus O_i\}$ (Minkowski difference)
4 merge any obstacles in $\{O'_i\}$ that have intersections
5 $V = \varnothing, E = \varnothing, C = \varnothing$
6 for $O'_i \in \{O'_i\}$ do
7     for $v \in O'_i$ do
8         if ISREFLEX($v$) then
9             $V = V \cup \{v\}$
10            $V_{visible}, A = $ VISIBLEQUERY($v, \{O'_i\}$)
11            $C$.insert($\{v, A\}$)
12            for $v' \in V_{visible}$ and ISREFLEX($v'$) do
13                if ISBITANGENT($v, v'$) then
14                    $e = (v, v')$
15                    $V = V \cup \{v'\}$, $E = E \cup \{e\}$
16 return $\mathcal{L}ayer(\mathcal{VG}(V, E), C)$
```

For example, Fig 4 shows parts of the Minkowski sums (black and brown solid lines) of the bounding polygon of the robot at two different poses (as dashed rectangles) that are 10 degrees apart with an obstacle (its red border is partially shown). In the figure, vertices A', B, and B' can rotate in both layers. To merge $\mathcal{VG}$s across the layers, for each vertex $v(x, y, [\theta_{lb}, \theta_{ub}])$ in the current layer, the connection with another vertex $v'(x', y', [\theta'_{lb}, \theta'_{ub}])$ from the neighboring layers will be established if: (1) v' falls into the visible area of v, (2) The edge (v, v') is bitangent, e.g., vertex A' and B as well as vertex B and B' in Fig 4 can be connected. However, if v and v' are directly connected by a line segment, the translation and rotation are coupled, meaning there is no guarantee of collision-free rotation at positions other than (x', y'). To address this, we decouple translation and rotation by introducing a substitute vertex $v''(x', y', [\theta_{lb}, \theta_{ub}])$ for v'. A two-fold connection between v and v' is then established by connecting (v, v'') and (v', v''), while ensuring that the total connection cost remains the same as the direct connection.

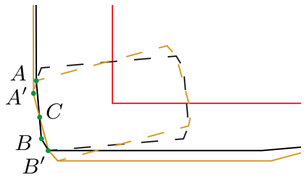

Fig. 4: Intersections of Minkowski sum boundaries (solid black and brown lines) of the robot at two poses (dashed lines) and an obstacle (partial border showing in red).

Such propagation can be divided into two passes - a *forward pass* and a *backward pass*. The *forward pass* iteratively propagates vertices from $\mathcal{L}ayer_i$ to $\mathcal{L}ayer_{i+1}$, and the *backward pass* propagates vertices from $\mathcal{L}ayer_{i+1}$ to $\mathcal{L}ayer_i$. In this way, each vertex can be propagated as far as possible. To avoid redundant propagation in the backward pass, we use a dictionary $D[\ldots]$ to cache information from the forward process. Each key is a layer index i, mapping to a set of tuples (i', j), where i' is the layer where the vertex was generated, and j is its index in that layer. In the *backward pass*, if $i' < i$, the vertex originates from a lower, already-checked layer, making further checking unnecessary.

Algorithm 3: VERTEXPROPAGATION(n $\mathcal{L}ayers$)

```
1 $V = \bigcup_{i=1\cdots n} V_i$, $E = \bigcup_{i=1\cdots n} E_i$, $C = \bigcup_{i=1\cdots n} C_i$
2 $D = \{1 : \{(1,0), \cdots, (1, |V_1|)\}, \cdots, n : \{(n,0), \cdots, (n, |V_n|)\}\}$
  ▷ The forward pass
3 for $l$ in $1 \cdots n$ do
4     $next = (l+1) \mod n$
5     for $(i, j)$ in $D[l]$ do
6         $v = V_i[j]$
7         for $(i', j')$ in $D[next]$ do
8             $v' = V_{i'}(j')$
9             if ISBITANGENT($v, v'$) and $v \in C[v']$ then
10                $v'' = (v.x, v.y, [\theta_{lb}, \theta_{ub}]_{(next)})$
11                $e_0 = (v, v''), e_1 = (v'', v')$
12                $V = V \cup v''$, $E = E \cup \{e_0, e_1\}$
13                $D[l] = D[l] \cup \{(i', j')\}$
14                $D[next] = D[next] \cup \{(i, j)\}$
  ▷ The backward pass
15 for $l$ in $n \cdots 1$ do
16    $prev = (l-1) \mod n$
17    for $(i, j)$ in $D[l]$ do
18        $v = V_i[j]$
19        for $(i', j')$ in $D[prev]$ do
20            if $i' < l$ then
21                continue
22            $v' = V_{i'}(j')$
23            if ISBITANGENT($v, v'$) and $v \in C[v']$ then
24                $v'' = (v.x, v.y, [\theta_{lb}, \theta_{ub}]_{(prev)})$
25                $e_0 = (v, v''), e_1 = (v'', v')$
26                $V = V \cup v''$, $E = E \cup \{e_0, e_1\}$
27                $D[l] = D[l] \cup \{(i', j')\}$
28                $D[prev] = D[prev] \cup \{(i, j)\}$
29 return $\mathcal{RVG}(V, E)$
```

C. Searching for Optimal Paths on $\mathcal{RVG}$

Given the start $s(x_s, y_s, [\theta^s_{lb}, \theta^s_{ub}])$ and the goal $g(x_g, y_g, [\theta^g_{lb}, \theta^g_{ub}])$, Alg. 4 is used to add s and g into the $\mathcal{RVG}$ by connecting the visible vertices in each layer that has overlap with $[\theta^s_{lb}, \theta^s_{ub}]$ and $[\theta^g_{lb}, \theta^g_{ub}]$. In practice, the start and goal generally have a fixed rotation, where we can simply set $\theta^s_{lb} = \theta^s_{ub}$ and $\theta^g_{lb} = \theta^g_{ub}$. Then, an optimal solution can be found according to the cost metric Eq. (1). During the search, each vertex v's rotation is represented by the mean value of its $[\theta_{lb}, \theta_{ub}]$. Since building $\mathcal{RVG}$ doesn't rely on the start and the goal, the same $\mathcal{RVG}$ can be used for multiple queries.

D. Formal Guarantees

Due to page limits, we state the key properties of RVG here. The full proofs and properties of RVG will be detailed in an extended journal version. Let n denote the resolution, m the total number of vertices/edges, and k denote the

Algorithm 4: ADDSTARTGOAL(s, g)

```
1 for $l$ in $1 \cdots n$ do
2     if $[\theta^s_{lb}, \theta^s_{ub}]$ overlaps with $[\theta^l_{lb}, \theta^l_{ub}]$ then
3         $V = V \cup \{s\}$ if $s$ not added
4         $V_{visible}, A =$ VISIBLEQUERY$(s, O_i)$
5         for $(i, j) \in D[l]$ do $v = V_i(j), E = E \cup \{(s, v)\}$;
6     if $[\theta^g_{lb}, \theta^g_{ub}]$ overlaps with $[\theta^l_{lb}, \theta^l_{ub}]$ then
7         $V = V \cup \{g\}$ if $g$ not added
8         $V_{visible}, A =$ VISIBLEQUERY$(q, O_i)$
9         for $(i, j) \in D[l]$ do $v = V_i(j), E = E \cup \{(g, v)\}$;
```

number of obstacles, including the obstacles and the robot. RVG has a $O(nm^2k)$ complexity.

Theorem III.1. *Assuming δ-clearance,* RVG *is resolution complete. Moreover,* RVG *computes asymptotically optimal solutions, if the overestimate of the rotation range converges to the true rotation range as the rotational resolution goes to infinity.*

Proof sketch. An optimal solution consists of connections between bitangent reflex vertices on the surface of obstacles in $SE(2)$, where there are infinite ways to connect the vertices if they have the same lowest cost. To see that RVG is asymptotically optimal, under the δ-clearance assumption, RVG builds an $\mathcal{RVG}$ that, at sufficiently high resolution, can approximate all the reflex vertices if the overestimate of the rotation range converges to the true rotation range. Any two bitangent reflex vertices can be connected in $\mathcal{RVG}$ with the equivalent cost. Resolution completeness follows asymptotic optimality. □

IV. COMPUTATIONL EVALUATION

We developed RVG with the hope that it would become a practical tool. As such, much effort has been devoted to efficiently implementing RVG in C++, to be open-sourced. RVG leverages CGAL [27] for two-dimensional geometrical computations and polygon operations. Computational evaluations were performed on an Intel i9-14900K CPU with a single CPU core/thread.

Qualitative evaluations are first provided, highlighting RVG's behavior as the resolution and the cost structure change. These qualitative results corroborate RVG's the correctness of RVG. Then, extensive quantitative evaluations, including full RVG performance characterization under different resolutions and performance comparisons with state-of-the-art sampling-based algorithms, confirm the superior performance of RVG, on both computational speed and solution quality fronts. Given limited pages, we mainly work with the cost metric where $\alpha = 1$ and $\beta = 0$ in Eq. (1). Note that even when we set $\beta = 0$, it is generally undesirable for the robot to execute unnecessary rotations because doing so will often add to the path's total length.

A. *Qualitative Behavior of* RVG

1) Effects of Varying Resolutions: RVG is an *anytime* algorithm, which finds a solution quickly. As the resolution increases, the solution quality improves. The experiment illustrated in Fig. 5 demonstrates RVG's such behavior. In the experiment, a rectangular robot is asked to go from the left to the right. At $n = 36$, the robot can only find a path above the large obstacle with $0.031s$ build time and $0.006s$ search time. As the resolution increases to $n = 72$, the robot finds a shorter path going through the narrower passage underneath the obstacle with $0.085s$ build time and $0.021s$ search time. The corresponding $\mathcal{RVG}$s (projected, without the orientation dimension) are also shown.

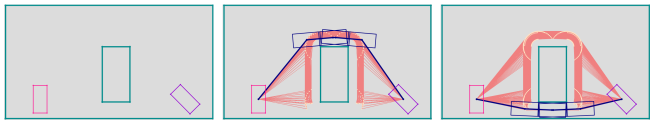
Fig. 5: The solutions from our algorithm with different resolutions. The pink, purple, and deep cyan rectangles represent the start configuration, goal configuration, and obstacles(including boundaries). [Left] The map. [Middle] The solution found at resolution $n = 36$. [Right] The solution at resolution $n = 72$, taking more time to compute.

2) Effects of Varying Cost Metrics: To be versatile, RVG should readily support different cost structures, allowing α and β in Eq. (1) to change freely. To showcase this capability, we carefully design the environment in Fig. 6. At resolution $n = 36$, results with four α and β combinations are given. On the top left, $\alpha = 1$ and $\beta = 0$. This forces the robot to select the shortest Euclidean distance path (in dark blue) of length 106.77 even though the path has a large rotation cost of 48.00 radians. When $\alpha = \beta = 0.5$ as on the top right, a path with a longer Euclidean distance of 116.34 is chosen, with a total rotation of 18.33 radians. The pattern continues. In the end, at $\alpha = 0$ and $\beta = 1$, the path with the least rotation (0.17 radian[1]) and the largest Euclidean distance (145.69) gets selected. On average, RVG takes $2.087s$ and $0.045s$ to build $\mathcal{RVG}$ and search for the optimal path. Qualitatively, RVG correctly computes the desired shortest path based on the provided cost metric.

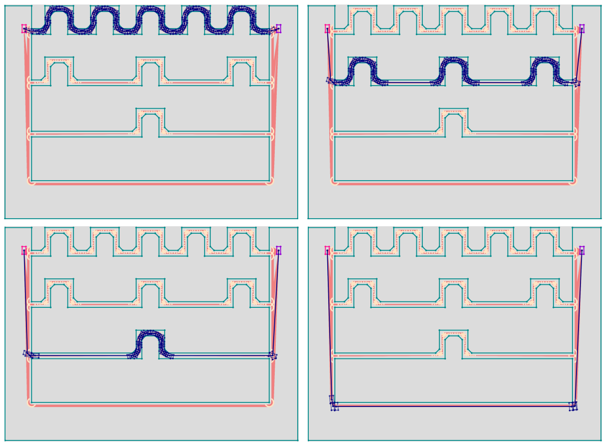
Fig. 6: The outcome of running RVG at resolution 36 with different α and β values. Both $\mathcal{RVG}$ and the shortest path (in blue) are shown. [Top Left] $\alpha = 1$ and $\beta = 0$. [Top Right] $\alpha = 0.5$ and $\beta = 0.5$. [Bottom Left] $\alpha = 0.48$ and $\beta = 0.52$. [Bottom Right] $\alpha = 0$ and $\beta = 1$.

[1]The actual rotation is zero; we are conservatively reporting the range of each orientation slice. In this case it is $2\pi/36 \approx 0.17$.

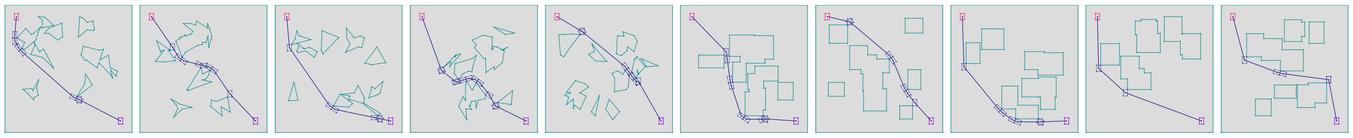
Fig. 7: The 10 randomly generated simple maps used in our evaluation, where we ask a rectangular robot to travel from the top left of the map to the bottom right of the map. For each map, the (Euclidean distance-based) shortest path, found at a resolution of 360, is also illustrated.

B. *Quantitative Performance of* RVG

For a fair comparison, we generate a set of random benchmarking problems similar to these used in [9], [13] for evaluating the performance of RVG, using the following procedure: Within predefined map boundaries, (x, y) positions are uniformly randomly sampled for placing convex polygonal obstacles. Each polygonal obstacle is also randomly created by sampling edges according to some proper rules. Intersecting polygons are merged.

Two example problems are illustrated in Fig. 8. The left subfigure shows the "simple" setting with a relatively small number of obstacles (total number of polygons sampled: 15). The right subfigure shows the "hard" setting with a large number of obstacles (total number of polygons sampled: 100). For both, the projected $\mathcal{RVG}$s at a resolution of 36 are shown, together with paths with the shortest distances as computed by RVG.

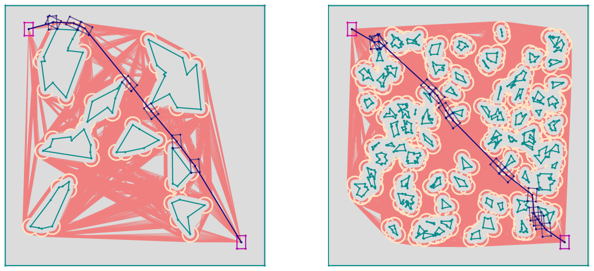
Fig. 8: Two randomly generated environments used in our performance evaluation. The maps, the $\mathcal{RVG}$s, the start and goal configurations, and the shortest paths found on the $\mathcal{RVG}$ are shown. [Left] A *simple* case. [Right] A *hard* case.

1) Performance of RVG*:* We begin by benchmarking the standalone performance of RVG on computational speed and solution quality. First, for simple maps, for reference, the 10 problems used for evaluation are shown in Fig. 7 together with the highest quality (Euclidean) shortest path found by RVG at a resolution of 360. We observe that the problems are rather diverse and already somewhat challenging (causing problems for sampling-based methods). In Fig. 9, the computational time for map building and search, as well as the path cost (length in this case since $\alpha = 1$) are shown for each map, for resolutions from 8 to 360.

We observe that building the $\mathcal{RVG}$ can be done well under a second at low resolutions, a few seconds at medium resolutions, and tens of seconds at high resolutions. Searching for the optimal solution on the $\mathcal{RVG}$ generally takes a fraction of a second except at the highest resolutions. For the majority of the randomly sampled evaluation test cases, most near-optimal paths (through manual examination of the map, and shown in Fig. 7) are found at low resolutions, generally taking a total of no more than 10 seconds.

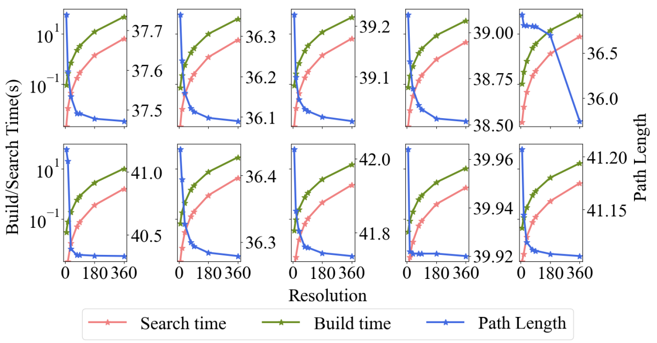

Fig. 9: Benchmark results for the 10 randomly generated maps given in Fig. 7. The $\mathcal{RVG}$ building time, path search time, and path length ($\alpha = 1$, $\beta = 0$ in Eq. (1)) are shown for resolutions $8, 18, 36, 72, 90, 180$, and 360.

In Fig. 10, benchmarking results similar to that in Fig. 9 are given for 10 randomly generated hard maps, one of which is given in the right subfigure of Fig. 8. The general trends here mimic those observed in Fig. 9, though the computation times now take longer. Again, we can compute a high-quality solution for most maps at low to medium resolution, taking tens of seconds.

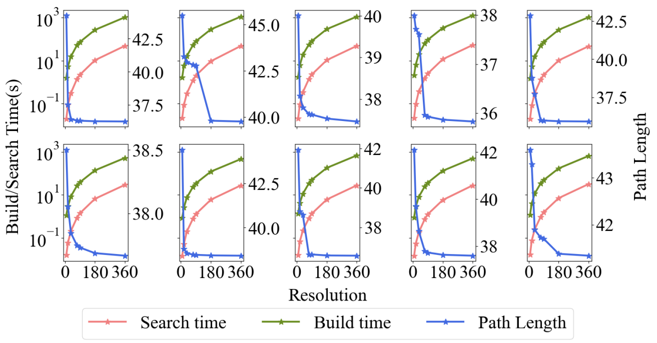

Fig. 10: Benchmark results for 10 randomly generated hard problems. The $\mathcal{RVG}$ building time, path search time, and path length ($\alpha = 1$, $\beta = 0$ in Eq. (1)) are shown for resolutions $8, 18, 36, 72, 90, 180$, and 360.

Fig. 12 presents a different view of the results provided in Fig. 9 and Fig. 10. Here, computation times and solution path lengths are scaled based on the times/lengths at the highest resolution 360, as the resolution varies. We note that a resolution of 360 is difficult to realize in practice, requiring the robot to move with no more than one degree of error. Therefore, solution costs obtained at this resolution can be regarded as the optimal solution in practice. Fig. 12 provides a clearer picture of when we hit the sweet spot in terms of good solution quality and fast computation. To reach a decent solution quality, e.g., around 1.02-optimal as compared with the solution quality at the highest resolution, a resolution of around 36 is generally sufficient.

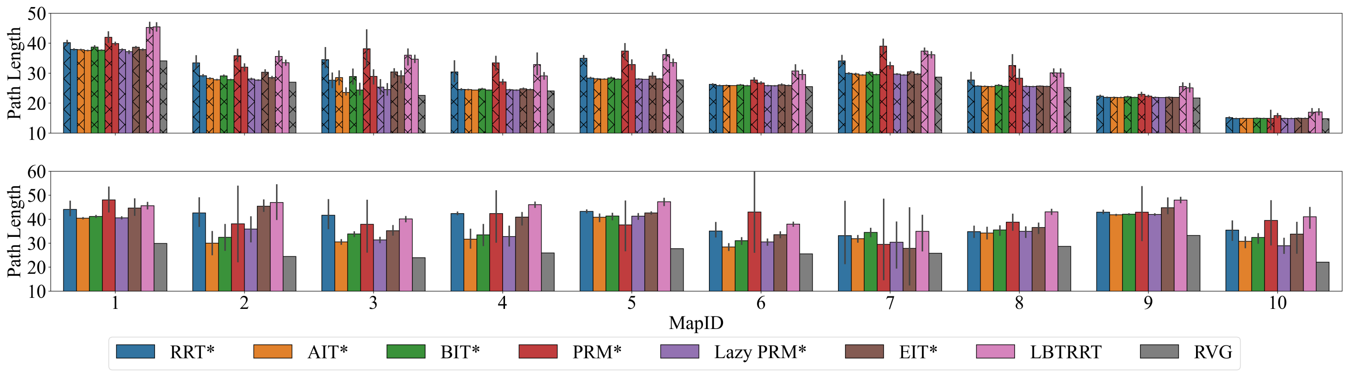

Fig. 11: [Top] Average path lengths over 10 runs for 10 environments returned by different motion planners with 1s/10s planning time vs RVG with 0.5s planning time on average (patterned/plain bars represent the results with 1s/10s running time). [Bottom] Average path length over 10 runs for 10 hard environments returned by different motion planners with 20s planning time versus RVG with 12.6s planning time on average.

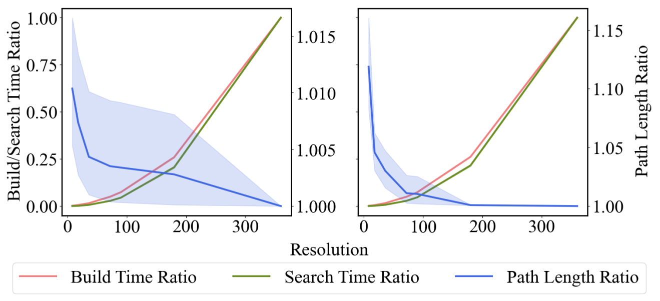

Fig. 12: $\mathcal{RVG}$ building time, path search time, and path length, scaled against these at the highest resolution. [Left] The ratios for the simple maps. [Right] The ratios for the hard maps.

2) *Comparison with State of the Art Sampling-Based Algorithms:* RVG, as a specialized multi-query method, shines even when compared with general single-query sampling-based planning methods. We use OMPL's implementations for all the other methods, and the collision checking is implemented with CGAL. Our first comparison (the top subfigure of Fig. 11) looks at the solution quality achieved for RRT*, BIT*, AIT*, EIT*, LBT-RRT, PRM*, LazyPRM*, and RVG where other methods use 1 or 10 seconds of planning time while RVG (at resolution 18) uses 0.94 seconds on average for these maps to build the $\mathcal{RVG}$ and to execute the search. In all cases, RVG outperforms the state-of-the-art single-query sampling-based methods even when these sampling-based methods are allocated ten times the planning time budget. If multiple queries are performed, then RVG's advantage will become more obvious. A similar comparison experiment is carried out for the hard maps with the resolution at 18 (the bottom subfigure of Fig. 11). Under the hard settings, we observe that the optimality gaps between RVG and the compared single-query methods widen. In Fig. 13, the following experiments are done on the simple maps. For each resolution of $8, 18, 36, 72, 90, 180$, and 360, we solve the instances with RVG and record the total map building and searching time. Then, the sampling-based algorithms are given the same amount of time to work on the same map. We then compare the resulting solution optimality using the path length computed by RVG as the baseline. We observe that RVG again consistently outperforms by a significant margin, even under the single-query setting. An analog for the hard maps is not provided, as performing the needed computation for the sampling-based methods will take hundreds of hours.

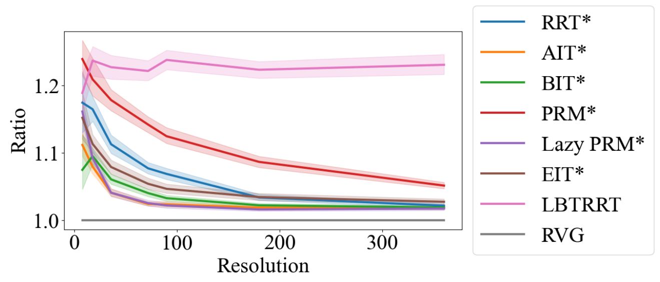

Fig. 13: Solution optimality ratio (y-axis) between sampling-based methods and RVG at different resolutions (x-axis) for simple maps, where each sampling-based method is executed for 10 times per data point with each run using the time budget used by RVG (including map building and path search) at the given resolution.

V. Conclusion and Discussions

In this study, we build the RVG algorithm to explore the natural idea of slicing the rotational degree of freedom, computing a 2D visibility graph for each slice, and connecting them to form a *rotation-stacked visibility graph* ($\mathcal{RVG}$), which can be subsequently used to carry out multiple approximate shortest-path queries for a polygonal holonomic robot traversing a 2D polygonal environment. After a $\mathcal{RVG}$ building phase, RVG supports path queries of different cost metrics as specified in Eq. (1). We establish that RVG is resolution complete and asymptotically optimal, and show that it achieves highly favorable results in comparison to state-of-the-art sampling-based methods.

Our study opens many follow-up questions; we mention a few here. First, only holonomic robots are currently supported. It would be interesting to explore extensions to $\mathcal{RVG}$ to support other types of robots, e.g., car-like robots [28]. Second, while RVG achieves decent performance, there is much room to improve its computational performance further. An obvious starting point to employ faster $\mathcal{VG}$ building and searching algorithms, e.g., adopting options mentioned in [29] and/or through parallelization of computation. As RVG matures further, it can potentially open up more applications, e.g., for autonomous driving.

References

[1] B. Huang, X. Zhang, and J. Yu, "Toward optimal tabletop rearrangement with multiple manipulation primitives," in *IEEE International Conference on Robotics and Automation*, 2024.

[2] F. Avnaim, J.-D. Boissonnat, and B. Faverjon, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE, 1988, pp. 1656–1661.

[3] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig, "Approximate motion planning and the complexity of the boundary of the union of simple geometric figures," in *Proceedings of the sixth annual symposium on Computational geometry*, 1990, pp. 281–289.

[4] D. Halperin and M. Sharir, "A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment," *Discrete & Computational Geometry*, vol. 16, no. 2, pp. 121–134, 1996.

[5] P. K. Agarwal, B. Aronov, and M. Sharir, "Motion planning for a convex polygon in a polygonal environment," *Discrete & Computational Geometry*, vol. 22, pp. 201–221, 1999.

[6] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[8] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[9] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3191–3198.

[10] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 521–528.

[11] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin, "Motion planning via manifold samples," *Algorithmica*, vol. 67, no. 4, pp. 547–565, 2013.

[12] O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality motion planning," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, 2016.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] L. E. Kavraki and S. M. LaValle, *Motion Planning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 109–131. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_6

[15] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Berlin, Germany, 2013, pp. 1–10.

[16] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization," *Computers & Chemical Engineering*, vol. 33, no. 3, pp. 575–582, 2009.

[17] K. Hauser, "Semi-infinite programming for trajectory optimization with non-convex obstacles," *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1106–1122, 2021.

[18] D. Zhang, C. Liang, X. Gao, K. Wu, and Z. Pan, "Provably robust semi-infinite program under collision constraints via subdivision," *arXiv preprint arXiv:2302.01135*, 2023.

[19] C. Liang, X. Gao, K. Wu, and Z. Pan, "Second-order convergent collision-constrained optimization-based planner," *IEEE Robotics and Automation Letters*, 2024.

[20] Z. Pan and Y. Zhu, "Provably feasible and stable white-box trajectory optimization," *arXiv preprint arXiv:2406.01763*, 2024.

[21] N. J. Nilsson, *A mobile automaton: An application of artificial intelligence techniques*. Stanford Research Institute Washington, DC, 1969.

[22] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[23] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[24] T. Lozano-Perez, *Spatial planning: A configuration space approach*. Springer, 1990.

[25] H. Hadwiger, "Minkowskische addition und subtraktion beliebiger punktmengen und die theoreme von erhard schmidt," *Mathematische Zeitschrift*, vol. 53, no. 3, pp. 210–218, 1950.

[26] F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller, "Efficient computation of visibility polygons," *arXiv preprint arXiv:1403.3905*, 2014.

[27] The CGAL Project, *CGAL User and Reference Manual*, 5.6.1 ed. CGAL Editorial Board, 2024. [Online]. Available: https://doc.cgal.org/5.6.1/Manual/packages.html

[28] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[29] J. S. Mitchell, "Shortest paths and networks," in *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017, pp. 811–848.